

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

Algoritmo MergeSort: cómo implementarlo en Python



brendomatos

10/06/2022



Anteriormente implementamos dos soluciones de búsqueda y en nuestro segundo enfoque, tuvimos que ordenar nuestra lista de estudiantes usando la función `sorted()` de Python. Pero imagina si no tuviéramos esa opción, ¿cómo podríamos ordenar la lista?

Primero, implementemos la clasificación de una manera más intuitiva. Para ello recorreremos la lista y para cada posición de esta iteración recorreremos el resto de la lista en busca del valor más pequeño. Si existe, intercambiaremos posiciones. Y llamaremos a esta solución `SelectionSort`:

```
from array import array
```

```
def ordenar(lista):
```

```
    tamaño_de_lista = len(lista) - 1
```

```
for posicion_actual in range(0, tamano_de_lista):
    posicion_menor = posicion_actual
    nombre_menor = lista[posicion_menor]

    for posicion_buscar in range(posiccion_actual, tamano_de_lista):
        nombre_buscar = lista[posicion_buscar + 1]

        if nombre_menor > nombre_buscar:
            mnombre_menor = nombre_buscar
            posicion_menor = posicion_buscar + 1

    if posicion_menor != posicion_actual:
        nombre_menor = lista[posicion_menor]
        lista[posicion_menor] = lista[posicion_actual]
        lista[posicion_actual] = nombre_menor

return lista

def main():
    lista_de_alumnos = ["Brendo", "Erica", "Monica", "Nico", "Paulo", "Rodrigo"]

    for nombre in lista_de_alumnos:
        print(nombre)

if __name__ == "__main__":
    main()
```

¡Todo funcionará perfectamente con nuestra lista corta de 7 estudiantes! Ahora, ¿qué tal si usamos la lista de aproximadamente 85,000 estudiantes? Mejor no, a no ser que podamos esperar mucho tiempo a la devolución.

Pero, ¿por qué el desempeño es tan malo para realizar esta pequeña operacion de orden?

Podemos ver que para cada estudiante en la lista, la revisamos nuevamente buscando el nombre más corto. Es decir, para ordenar una lista de N alumnos, realizamos n^2 operaciones y en nuestro caso $(85000^2) = \mathbf{7.225.000.000}$ **operaciones**, eso sí, más de 7 billones de operaciones.

Así que consideramos este algoritmo como **$O(n^2)$** un algoritmo cuadrático. En este artículo hablamos más sobre la notación Big O.

Entonces, ¿vamos a implementar otra solución de clasificación más eficiente?

Volvamos una vez más a las técnicas de divide y vencerás para resolver el problema de encontrar el nombre más corto en la lista.

En la función de clasificación, llamemos a la función `merge_sort()`, que toma como parámetros:

- La lista de estudiantes;
- Una lista temporal con el tamaño predeterminado de la lista de estudiantes;
- El índice inicial;
- El índice final de la lista.

Y se realizarán los siguientes pasos:

- Dividiremos la lista en dos y llamaremos recursivamente a la función `merge_sort()`, pasando como parámetros:
- Los datos de la primera lista;
- Los datos de la segunda lista,

Finalmente, llamaremos a la función `merge()` para fusionar las dos listas ordenadas con el apoyo de la lista temporal, reescribiendo la lista original.

```
from array import array
```

```
def importar_lista(archivo):  
    lista = []  
    with open(archivo) as tf:  
        lines = tf.read().split('"', '"')
```

```
for line in lines:
    lista.append(line)
return lista
```

```
def ordenar(lista):
    tamaño_de_lista = len(lista)
    lista_temporaria = [0] * tamaño_de_lista
    merge_sort(lista, lista_temporaria, 0, tamaño_de_lista - 1)
```

```
def merge_sort(lista, lista_temporaria, inicio, fin):
    if inicio < fin:
        medio = (inicio + fin) // 2
        merge_sort(lista, lista_temporaria, inicio, medio)

        merge_sort(lista, lista_temporaria, medio + 1, fin)

        merge(lista, lista_temporaria, inicio, medio + 1, fin)
```

```
def merge(lista, lista_temporaria, inicio, medio, fin):
    fin_primera_parte = medio - 1
    indice_temporario = inicio
    tamaño_de_lista = fin - inicio + 1

    while inicio <= fin_primera_parte and medio <= fin:
        if lista[inicio] <= lista[medio]:
            lista_temporaria[indice_temporario] = lista[inicio]
            inicio += 1
        else:
            lista_temporaria[indice_temporario] = lista[medio]
            medio += 1
        indice_temporario += 1

    while inicio <= fin_primera_parte:
        lista_temporaria[indice_temporario] = lista[inicio]
        indice_temporario += 1
        inicio += 1
```

```
while medio <= fim:
    lista_temporaria[indice_temporario] = lista[medio]
    indice_temporario += 1
    medio += 1

for i in range(0, tamanho_de_lista):
    lista[fin] = lista_temporaria[fin]
    fin -= 1

def main():
    lista_de_alunos = importa_lista('../data/lista_alunos')

    ordenar(lista_de_alunos)

for nombre in lista_de_alunos:
    print(nombre)

if __name__ == "__main__":
    main()
```

Al ejecutar el algoritmo vemos que el rendimiento fue mucho mejor. ¿Entendemos lo que pasó?

Cuando partimos la lista por la mitad y la ejecutamos una y otra vez para realizar las comparaciones, nos recuerda mucho al algoritmo de búsqueda binaria, ¿verdad? ¡Sí! Entonces, por ahora, consideremos este algoritmo como **$O(\lg N)$** .

También tendremos en cuenta que realizamos la fusión que pasa por las dos mitades de la lista para comparar y llenar la lista temporal de manera ordenada. Entonces este algoritmo es lineal, es decir, **$O(N)$** . Como ambos algoritmos se ejecutan juntos, podemos considerar el **MergeSort $O(N \lg N)$** .

De todos modos, vimos que el rendimiento de MergeSort es muy superior a SelectionSort, pero ¿es esta nuestra mejor solución? ¿Podemos resolver las clases de otras maneras?

Y la respuesta es: hay numerosas formas de resolver los problemas de clasificación y en el próximo artículo, en la serie trabajaremos en una forma más de resolver el problema de

clasificación.

Brendo Rodrigo Souza de Matos

Ingeniero de Software apasionado por lo que hace, amante de los nuevos retos y sediento de conocimiento. Actualmente soy Ingeniero de Software de Plataformas en Méliuz (B3: CASH3) y estoy realizando una Maestría en Ciencias de la Computación en la Universidad Federal de Amazonas.

Este artículo fue adecuado para Alura Latam por: [Wilfredo Rojas](#)

Cursos de Programación

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

**En Alura encontrarás variados cursos sobre Programación.
¡Comienza ahora!**

SEMESTRAL

US\$49,90

un solo pago de US\$49,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana

- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

[Paga en moneda local en los siguientes países](#)

ANUAL

US\$79,90

un solo pago de US\$79,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas

- ✓ Acceso a todo el contenido de la plataforma por 12 meses

¡QUIERO EMPEZAR A ESTUDIAR!

[Paga en moneda local en los siguientes países](#)

Acceso a todos
los cursos

Estudia las 24 horas,
dónde y cuándo quieras

Nuevos cursos
cada semana

NAVEGACIÓN

PLANES
INSTRUCTORES
BLOG
POLÍTICA DE PRIVACIDAD
TÉRMINOS DE USO
SOBRE NOSOTROS
PREGUNTAS FRECUENTES

¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

BLOG

PROGRAMACIÓN
FRONT END
DATA SCIENCE
INNOVACIÓN Y GESTIÓN
DEVOPS

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES



ALIADOS



En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

CURSOS

Cursos de Programación

Lógica de Programación | Java

Cursos de Front End

HTML y CSS | JavaScript | React

Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps

Docker | Linux

Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento